Data Acquisition and Control Software for AMANDA's String Eighteen

John Jacobsen

jacobsen@rust.lbl.gov

Lawrence Berkeley National Laboratory

Version 1.9
May 24, 2002

Look for updated versions of this document at
`http://rust.lbl.gov/~jacobsen/docs/string18_software.pdfand`
`http://rust.lbl.gov/~jacobsen/docs/string18_software.doc`

CONTENTS

1 Introduction

Two kilometers below the surface of the ice covering the South Pole, a set of 677 optical sensors operates continuously, collecting very faint flashes of light from muons and neutrinos. This instrument, known as the Antarctic Muon and Neutrino Detector Array (AMANDA), is the largest existing detector of high-energy cosmic neutrinos. Neutrinos are elusive particles which can carry information about distant astronomical objects. Because they have little mass and no charge, they can travel directly to earth from distant objects. This makes them a useful tool for astronomy. The fact that they pass so readily through matter means both that they are very difficult to detect and that they can convey information from places that might be hidden by intervening matter. Neutrinos are also signatures of some of the most energetic processes in the universe.

The basic design concept of AMANDA is as follows. Interactions of neutrinos with atoms of ice generate energetic muons, which in turn radiate faint flashes of light as they travel through the ice. Some of this light is captured with very accurate time resolution by optical sensors, deployed in long "strings" in the ice. The time of arrival of the photons allow one to reconstruct the direction of motion of the muon, and therefore of the neutrino. This direction gives the point in the sky where the neutrino came from. Accurate time resolution is the key to doing astronomy with AMANDA.

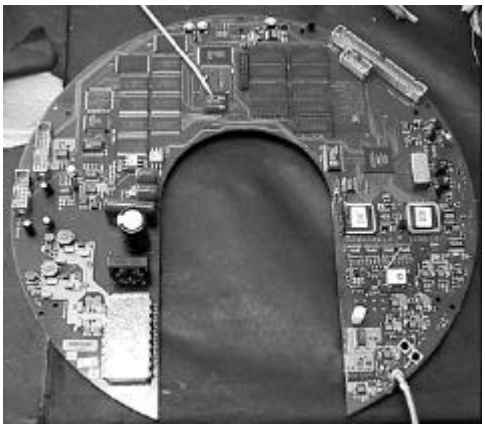


Figure 1 - Printed Circuit Board from one of the String 18 Digital Optical Modules (DOMs)

Most of the sensors in AMANDA work by converting photons into electrical signals, then into brief, intense pulses of light which travel from the sensor to the surface along a long fiber optic cable. In this design, power is sent from the surface to the sensor via a separate electrical cable. Fiber is superior to electrical for transmission of the pulses of data, because of the dispersion of electrical cables which degrades timing resolution. But this design requires two cables which represent extra cost and added possible failure modes. An alternative is to digitize the pulse from the light sensor in an embedded computer *before* transmission, and then to transmit the pulse to the surface in a digitized form which is less vulnerable to the dispersion effects of the cable. This removes the necessity of the fiber optic cable.

AMANDA's 18th string (of 19) consists of forty modules using this design concept, called the Digital Optical Module (DOM). These modules were deployed in early 2000, in order to test the technology with an eye to the next generation detector, known as IceCube. The modules consist of a photomultiplier tube (PMT), which detects the photons and turns them into electrical pulses; various amplification and digitization electronics; a programmable FPGA which contains much of the logic required to operate the sensor; and an ARM CPU for handling the digital communications and servicing requests from the surface. There is also volatile and non-volatile memory for storing physics data, FPGA firmware and programs which run on the CPU.

At the surface, the cables from the DOMs are attached to 40 custom communications cards (DOMCOM cards) in five industrial PCs (DOMCOM PCs). The DOMCOM cards can be thought of as specialized serial ports with functions for powering on and off each DOM, and sending and receiving time calibration pulses. The five DOMCOM PCs are networked together on 100BaseT switched Ethernet, along with a

sixth master control PC. This network is then accessible via the station LAN and, at certain times during the day, to the outside world via satellite connection.

2 Overview of the Software

This document describes LBNL's contribution to the software which runs in the DOMCOM PCs and the control PC. The principle job of the software is to allow one to control and communicate with the DOMs in the ice, allowing one to collect as much data as possible from the optical sensors. The additional task of applying time calibrations to this data, and building space-time coincidences corresponding to actual particles, was left to collaborators at the University of Pennsylvania. As of this writing, this functionality has not been fully implemented (see Section 4.3.5 for more information).

The software was implemented mostly by the author (Jacobsen), with some specific code contributions by Chuck McParland and Azriel Goldschmidt. The DOMCOM hardware design came from K. Sulanke at DESY/Zeuthen, who built upon a test design by J. Ludvig and G. Przybylski at LBNL. All three groups (Penn, DESY and LBNL) contributed to the design of the DAQ system as a whole.

This document does not present an exhaustive overview of all the features of the software or the technical intricacies involved with its implementation, but it should present a working introduction to the conceptual framework of the software architecture as well as basic hands-on information on how to install and use the software. The software itself is somewhat liberally commented and can be referred to if extra details are needed (or you can contact the author at jacobsen@rust.lbl.gov).

2.1 Software Organization

The software described here can organized into three levels of abstraction. At the lowest level, closet to the DOMCOM hardware, is a device driver running in each DOMCOM PC. This driver, operating as a dynamically-loadable Linux kernel module, allows data transmission to and from the DOM to be treated like file input and output; writing to the device file for a particular DOM causes data to be sent from the DOMCOM card to its respective DOM; data sent from the DOM to the DOMCOM can be read from the device file as if it were a normal data file.

At the next higher level, the **domserver** program uses the device driver and system network functions to connect the DOMs to programs running on other PCs, so that data from each DOM can be collected by programs running anywhere on the network.

At the highest level, an executive program ("**domexec**"), interacts with domserver on each of the five DOMCOM PCs, and allows the operator of the experiment to switch the detector on and off, send the appropriate control parameters to each DOM, and begin or end data taking runs.

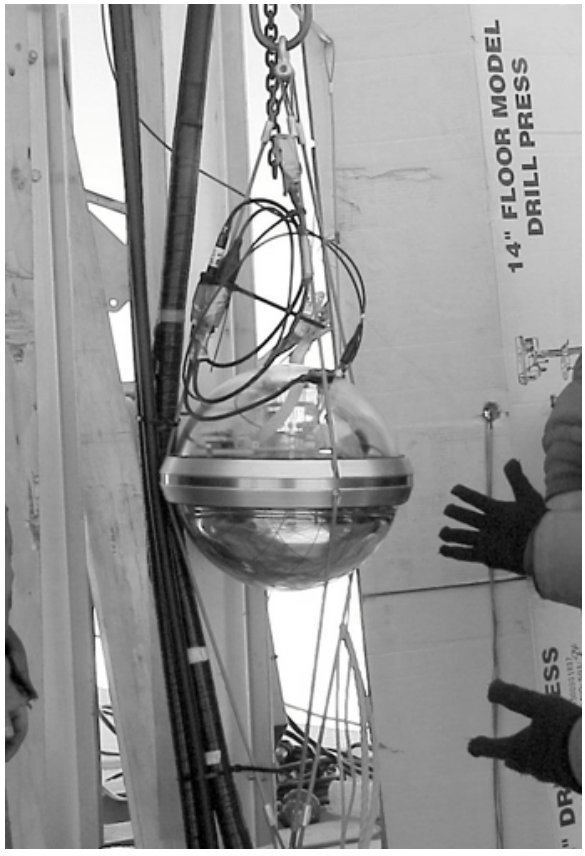


Figure 2 - Fully assembled DOM being lowered into the ice at the Pole

Several other programs (e.g., **domtest, domtalk, domcom**) have functions for the configuration and testing of various system components.

Not covered in this document are two programs being developed by collaborators from Penn, namely RAPCal and EBTrig, which are meant to consume the PMT data collected by domserver, apply time calibrations to this data and select light signals from different DOMs grouped closely in time to form "triggers" corresponding to physical events (particles passing through the detector).

Also not fully described here are the embedded software inside the DOMs: the DOM boot program (**domboot**) and the DOM Application, which were created at LBNL by D. Lowder and C. McParland, respectively.

2.2 DEVELOPMENT ENVIRONMENT

All the software in this document was built using Open Source tools, most of which are packaged with standard Red Hat 7.x Linux distributions. The bulk of the programming is done in C using the GCC compiler, with some programs written in Perl and requiring external Perl modules available from the CPAN archive.

2.3 SOURCE CODE VERSION CONTROL

The software was developed and maintained in a Concurrent Versions System (CVS) archive at LBNL, with copies at the South Pole. This greatly facilitated software installation at the remote site and reduced version conflicts between different copies of the software. The master tree at LBNL is called **domsoft** and lives on the machine rust.lbl.gov.

An account is needed on rust to check out the domsoft distribution. For more information, see Section 5, *Acquiring and Building the Software*.

Figure 3 - String 18 Electronics at South Pole, showing the five DOMCOM PCs (tbdaq-1 through -5) with the control PC just below. (Picture by K. H. Sulanke)

2.4 USER INTERFACE PHILOSOPHY

Most of the programs described here are system utilities that normally don't require interaction with a user. However, the highest level programs such as domexec and domtest are written with a user / operator in mind. These programs are text-based rather than GUI-based. This choice was based on the realities of low bandwidth communications to and from the South Pole. It has allowed for extensive remote operation of the DOMs from the northern hemisphere, enabling us to demonstrate features and performance of the digital system during the Winter seasons when *in situ* intervention in the system would be difficult or impossible. It also has the advantage of relative platform-independence, requiring only a secure-shell connection to handle all operations.

The user typically has two ways of interacting with these programs. The simplest is to type the name of the command at the Unix shell prompt, and follow the menu choices by typing the appropriate key. The other is to specify which action to take on the command line using various switches and arguments. In general, the "-h" switch tells you the available options, e.g.,

```
$$ domtalk -h

/usr/local/dom/bin/domtalk version V0.2 : A Linux / Perl Program to interact
     with DOM Boot codes
by John Jacobsen at LBNL.

Usage: /usr/local/dom/bin/domtalk [terminal_server] [port_num|test_board_id]
          [-l logfile]
          [-s "send this string to term_server and quit"]
          [-t time] (Time in msec between characters sent)
          [-e script] Execute script on DOM and exit
          [-da application] Download "application" to DOM
          [-an application_name] Use application_name for flash file sys
          [-df fpga] Download firmware file "fpga" to DOM
          [-fn fpga_name] Use fpga_name for flash file system
          [-p] Set the preferences in the flash (use with -an/-fn  options)
          If port_num < 8, port of 4000 + test_board_id is used.
```

Specific details on the use of domtalk and other programs can be found in Section 4.

3 History and Evolution of the String 18 Software Effort

The String 18 software evolved in parallel with the hardware development and production, and in a somewhat organic fashion. The first interactions with prototype DOM boards occurred during 1999 using a freeware program called TeraTerm running on Windows NT platforms. These sessions relied on a standard serial cable connection between the DOM board and the user's PC.

More functionality quickly became necessary, such as the ability to download files into flash memory on the DOM boards, and to send associated CRC information as a quality check. The Perl program called **domtalk** was written in late 1999 to address these needs. The DOMs were designed to run a low-level boot program, called **domboot**, at power-up, during which the user could interact with the DOM via a simple shell interface. Domtalk provided a simple interface to domboot via the serial port, much like telnet for the connection of ordinary computers on a network. Perl was chosen at this early date because of the large support base in the form of external modules (CPAN), and because of the high speed with which code could be written to do relatively straightforward things. Network accessibility was added by connecting the DOM boards to Lantronix terminal servers via the serial interface; domtalk could then be run on any computer with a TCP/IP network connection. Domtalk relied on a lower-level Perl module (**GenericDOMSerial**) which provided a general means for sending data to and from DOMs. While originally both WinNT and Linux were supported in this interface, and both serial and network modes of access were allowed, the interface was eventually simplified (**LinuxDOMSerial**) to support only networking and Linux (though any straightforward Unix distribution should still work).

Another program, **domtest**, was written in late 1999 to communicate with DOMs whose boot programs had bootstrapped into the **DOM Application**. When running in application mode, the DOMs require messages to be sent in a certain packetized format, rather than the free-form text format used by domtalk and domboot. Domtest uses this message format to communicate with the application. It uses custom Perl modules built on top of LinuxDOMSerial to implement this packetization scheme in a straightforward way (see Section 4.2.3, *Messaging*, for more information on the message format).

The forty-one String 18 modules were deployed in early 2000, and domtalk and domtest were used extensively in the commissioning of the first version of the surface electronics. Domtest was expanded to use Perl DBM databases to store and retrieve high voltage settings and other information for each DOM. Another program, **domlogger**, was written to collect sample data from the DOMs for daily transmission to the northern hemisphere from the Pole.

Domlogger ran nearly uninterrupted for the Austral Winter season of 2000 and provided a large data sample for study. In early 2001, under serious budgetary constraints, another surface electronics prototype was deployed at the Pole, allowing for the concurrent, full time calibration of four DOMs. Domlogger was enhanced to record this data, and a C program called **syncserver** was written to implement the time calibration functions on the surface. Time calibration and muon detection were successfully achieved; on the basis of this success, the Digital Optical Module was chosen as the technology to be used for the sensors of IceCube. At the same time, preparations were begun to instrument the entire string (all 41 sensors) with fully-functional surface electronics. This is the DOMCOM hardware for which the software described in this document was written. The new hardware was successfully deployed in the 2001-2002 Austral Summer season, and remotely tested and operated via satellite in 2002.

The new hardware brought the possibility of collecting data in real time from 41 sensors at much higher data rates than previously available. For this task, the domlogger program, written in Perl, was too slow and unwieldy. The terminal servers were eliminated and a device driver was written to allow the fast buffering of communications data from the DOM. The **domserver** program was written in C which allowed programs running on any computer on the network to interact with the DOMs and consume the data they would produce. As a partner to domserver, the executive program **domexec** lets a user control any number of DOMs on multiple DOMCOM PCs, to begin and end data collection runs. In the current version of the software, domtest and domtalk are still used for testing and configuration of each channel. These programs could be incorporated into the executive for more flexibility. In general, however, the domexec/domserver framework is a reasonable prototype of the projected design for the greatly expanded IceCube project, with domserver running on the DOMCOM PC providing a model for the IceCube DOM Hub.

4 The Software in Detail

In the next sections, the various components of the current software are explained in some detail. We start with the lowest level, the device driver, because its role is conceptually the simplest. In the next subsection, the key programs domserver and domexec are treated together because they interact so closely. In the last subsection, various additional programs are described. The Appendix shows a diagram of the various components in the system and may help elucidate the concepts presented in the text.

4.1 THE DOMCOM DEVICE DRIVER

The basic task of the String 18 data acquisition system is to provide communications to the DOMs, to control them and retrieve data from them. The simplest problem in the software architecture, from a conceptual standpoint, is to create a means for sending and receiving bytes to and from the DOMs.

The "tb" device driver provides a low-level (byte-wise) communications interface to the DOMs via the DOMCOM boards. FPGA logic in the DOMCOM boards provides input and output FIFOs connected to UARTs in the DOMCOM boards. The UARTs in turn drive the communications circuits. By writing to and reading from FPGA registers mapped to port addresses (e.g. 0x0300), the tb driver allows Linux programs to communicate with the DOMs.

The driver is meant to be used in conjunction with domserver (described in Section 4.2), which connects the DOMCOM devices to network sockets to make them available on the network in various ways. In the most direct of these mechanisms, data written to a particular domserver socket gets sent to the DOM, and data received from the DOM gets sent to the remote program over the same socket. Domserver can also interact directly with the DOMs through the tb driver, specifically when a run is in progress and messages are exchanged to pull data from the modules; this data is only sent over the network if a "consumer" program such as RAPCal or SimRAPCal is connected.

Domserver interacts with the DOMs both by communicating via characters, and by carrying out time calibrations. The latter operation requires steps in which specialized commands are issued by domserver to

the DOMCOM FPGAs to generate timing pulses to the DOMs, and to read out digitized return pulses from the DOMs. These FPGA commands, implemented using the portio.c interface in domsoft/src/portio, are independent of the tb device driver. This slightly unconventional approach is mostly for historical reasons (a more standard way would be to implement these functions as ioctl() calls), and also because the communications functions provided by the tb driver are more or less independent of the other DOMCOM board functions handled with the portio interface (though care must be taken not to send timing pulses during normal communications).

4.1.1 How the Driver Works

The tb driver is a Linux kernel module. That means first of all that it runs with kernel privileges (to allow it access to the low level hardware, and to run quickly "at interrupt time" when needed). It also means that it can be loaded into the kernel dynamically, after the system has booted, and unloaded later at any time. This greatly enhances the ease of driver development and testing, because it eliminates the need to reboot the system when trying a new version of the driver.

The driver can be thought of as having two parts - an interrupt handler, to allow for the fast buffering of incoming data from the DOMCOM board, and a system interface, which consists of open, close, read and write functions that allow the device to be treated like a file.

There are eight possible DOMCOM board IDs (0-7), configurable on the hardware by setting DIP switches. Each DOMCOM board is addressed through the ISA bus based on its DOMCOM board ID.

4.1.1.1 Device files, major and minor numbers

The device files for the driver are /dev/tb0, … , /dev/tb7. The major number is 88 (defined in domsoft/driver/tb.h) and the minor number is the DOMCOM board ID. When a user program opens one of the device files, it causes the driver to initialize buffer space for that DOMCOM board, enables interrupts on that board, and enables the hardware UART (so that communications are routed through the firmware UART on the DOMCOM board, rather than through the RS-232 connection to the optional terminal server hardware). It also clears the input FIFO in case any garbage data is waiting to be read out.

Each of the device files /dev/tb* can only be opened by one process at a time.

4.1.1.2 Interrupts and circular buffers

Interrupt request (IRQ) number 10 is used. This can be changed in domsoft/portio/portio.h. If the IRQ is changed, the DOMCOM PC BIOS must be changed to handle "legacy ISA" for that interrupt line[1]. When this interrupt is issued by the FPGA of one of the DOMCOM boards, the interrupt handler is called, and all the DOMCOM boards with data are read out into circular buffers (1024 kB for each DOMCOM board). To guard against race conditions, these circular buffers have pointers for "producer" (interrupt handler) and "consumer" (read function) ends. They also have a counter which indicates how many times the buffer has been wrapped (i.e., how many times the pointer has gone off the end and put at the first byte) for both the producer and consumer. This guarantees that any overflows of the buffer are caught. (If a buffer overflow occurs, the next read will fail, and a message will appear in the system log /var/log/messages.)

4.1.1.3 Read and write

After a user program opens one of the files /dev/tb[0,…,7], it can then read() from it or write() to it. The read() system function causes the driver function read_tb() to be called, which pulls the appropriate number of bytes from the circular buffer of the corresponding DOMCOM board. Similarly, write() causes write_tb() to be called, which writes data to the correct FPGA registers so that data appears in the output FIFO of the desired DOMCOM board.

[1] Refer to J. Jacobsen, "DOMCOM PC Installation Instructions." See Bibliography.

Only non-blocking reads are currently allowed (many devices cause a reading application to halt execution, or "block," until data is available to be read, but since some programs need to read from both the connecting socket and the hardware, this functionality is generally not wanted). Select() or poll() are needed to determine if data is available to be read. Write also won't block but may return only partially completed if the DOMCOM board's output FIFO is full. In that case, a subsequent call to write will be needed. NOTE: if calling write repeatedly, put a usleep of ~10 msec in; if you simply busy-wait by calling write() repeatedly, large amounts of CPU time will be wasted.

4.1.2 Diagnostics

The device driver uses the system log for diagnostic output (as does domserver). When opening or closing the DOMCOM board device, or when errors occur, the tb driver causes messages to be written to the system log (/var/log/messages). A window on the DOMCOM board control PC running "tail -f /var/log/messages" will show these messages as they are generated. Additional diagnostic messages are also available in the driver, but you will have to change the code for those statements, substituting "printk" for "pprintk" and recompiling (see below).

4.1.3 Installing and Running the Driver

4.1.3.1 How to check out the driver code

The driver code is part of the DOM software archive (domsoft), organized using the CVS system. The subdirectory src/driver contains the bulk of the device driver code, although the src/portio subdirectory contains hardware information used by both the driver and domserver. The src/init subdirectory contains a startup script for /etc/init.d which installs the driver at boot time. (See Section 5.1, *How to use the domsoft Repository*.)

4.1.3.2 How to compile the driver

Change to the driver subdirectory (cd domsoft/src/driver). Compile by typing "make".

4.1.3.3 How to install the driver

The driver should already be installed on the DOMCOM PCs at LBNL and at the Pole. The **tbrc** startup script runs automatically at system boot time to install the release version of the driver (in /usr/local/dom/driver), and also to run domserver.

To install the driver by hand: become root. Change to the driver subdirectory. Make (should do nothing if already compiled). Install in release directory by typing "make install". If you want to load the driver by hand in the currently running kernel, "insmod tb.o". To remove the driver, "rmmod tb". You will have to do this first before reloading if the driver is already running. If any process is using the driver (i.e. has one or more of /dev/tb* open), you won't be able to remove the driver.

To see if the driver is installed, "lsmod".

4.1.4 Troubleshooting

The best way to test the driver is to start trying to talk through it. Install it, start domserver running on the DOMCOM PC (we'll use tbdaq-6.lbl.gov for this example), and start a tail of /var/log/messages. Run domtalk to that PC (tbdaq-6) and the appropriate channel number (0-7), and power-cycle the DOMs. If the driver is running, you should see the DOM boot prompt appear:

```
Welcome to DOMBOOT release (CRC Enabled) of 15-Nov-1999
CRC table initialized…
Timeout value set to 30 seconds…
```

After the return key is pressed, the domboot prompt should appear:

```
Domboot 1.16 DOM 16  Enter command (? for menu):
```

You can also look at the system log (/var/log/messages) and see some of the driver output, both when the kernel module is installed, and when a connection is established by domtalk.

If there are problems, the following checklist should also help make sure everything is in place:

? DOM cabled properly to DOMCOM Board?
? Power supply connected, and voltage set to 80 V?
? Device driver loaded? (lsmod to check)
? Domserver running? (ps ax | grep domserver to check)
? DOMTalk running, connected to correct server and port?
? DOMCOM board FPGA loaded? (use program "domcom" to reload, if in doubt; see Section 4.3.4)

If these things are in place, you should see the domboot prompt in domtalk when you cycle the power on the DOMs (if the DOM is in application mode, power-cycling will return to boot mode and show the boot prompt).

4.1.5 Additional Documentation

For details on the functions provided by the DOMCOM FPGA design, please refer to K. H. Sulanke's DOMCOM FPGA API document and the additional document by G. Przybylski's (see Bibliography).

4.2 THE CORE SOFTWARE - DOMSERVER AND DOMEXEC

4.2.1 Introduction

A key concept in the design of both String 18 and IceCube is to provide a network connection point (a DOMCOM PC or DOMHub) for a set of DOMs, and a program for interacting with a number of these connection points to control the system as a whole. In essence, it is a client/server model, with the server being the DOMCOM or DOMHub and the client being an executive program which can connect to these systems and cause them to direct the DOMs to do various things.

For String 18, these tasks are implemented respectively in **domserver** and **domexec**. Domserver runs on each DOMCOM PC and talks to up to 8 DOMs; domexec runs on the string control PC called *string18*, and talks to domserver running on each of the five DOMCOM PCs.

For example, to start a run, a user logs into string18 and runs domexec, selecting the appropriate menu option or giving the command-line shortcut. Domexec connects to multiple ports on each of the five DOMCOM PCs and tells each copy of domserver to put its DOMs in the running state.

This system design is very modular, in that one can create a test system with a single PC, run both domexec and domserver on that PC, and have the same functionality as with the full system. One can add additional DOMCOMs without any change to the software design.

4.2.2 Functional Layers

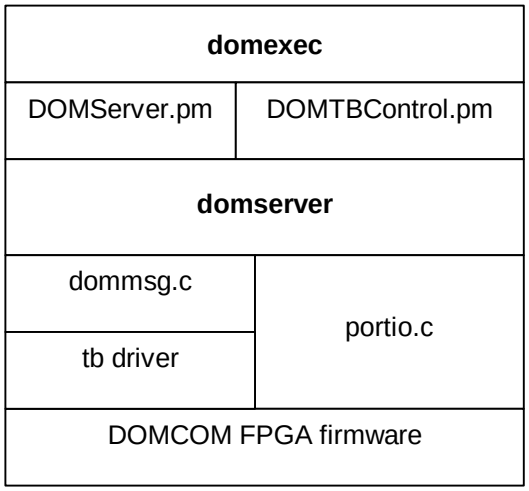
The software design relies on a set of layers of abstraction which go progressively farther from the hardware and closer to what the user sees.

At the top layer, domexec (written in Perl) relies on the **DOMServer**.pm and **DOMTBControl**.pm modules for access to domserver via network sockets.

The C program **domserver** receives single-byte commands sent over the network via these Perl modules, along with any argument data that might be required. Domserver handles these requests by either sending messages to the DOMs using functions from **dommsg.c**, or by calling functions which interact with the DOMCOM boards using **portio.c**. Domserver then replies with status information and any requested response data.

The dommsg functions use the device driver to communicate with the DOMs; the portio functions use read and write operations to DOMCOM FPGA registers to accomplish tasks such as reading timestamps, sending timing waveforms to the DOMs, and other functions.

Hierarchy of Functional Layers



4.2.3 Messaging

As mentioned above, normal operations of the DOMs involve passing messages in a well-defined format between domserver, running in the DOMCOMs on the surface, and the DOM Application, running in the DOMs in the ice. These transactions are always initiated by a message from the surface, with the DOM replying with its own message.

The messages consist of three or more packets. These packets are 10 or more bytes in length, with a start byte identifying the packet type, and the last byte a carriage return (CR, or ASCII 13). Normally there are eight bytes between the start byte and the terminator byte, although this number can be as large as 16 using the following escape mechanism, which prevents CR from appearing anywhere in the packet data except at the end: define ESC to be ASCII 33; CR then becomes ESC followed by DEL (ASCII 127), and ESC becomes ESC ESC. This mapping is reversed in the decoding of packets on the other end.

The packets in a message occur in the following order:

BEGIN_MESSAGE	Beginning of message packet
MESSAGE_HEADER	Message header packet
[MESSAGE_DATA]	Message data packet (optional)
.	
.	
.	
END_MESSAGE	End of message packet

The format of each packet (before the ESC-ESC and ESC-DEL chicanery) is as follows:

BEGIN_MESSAGE:
Byte 0: Begin message marker: FL_BEG_MSG (0x02)
Byte 1: Message ID
Byte 2: 0
Byte 3, 4: Number of packets in the message (big-endian)
Bytes 5-8: 32-bit CRC value for the data, if any
Byte 9: CR

MESSAGE_HEADER:
Byte 0: Message marker: FL_MSG (0x03)
Byte 1: Message type (thread in the DOM which should handle / did handle request, e.g. "slow control")
Byte 2: Message subtype (particular operation which was requested, e.g. "read an ADC value")
Bytes 3, 4: Length of data to DOM (big-endian)
Bytes 5,6: Reserved
Byte 7: Message ID
Byte 8: Status of message (if message is response from DOM)
Byte 9: CR

MESSAGE_DATA:
Byte 0: Message marker: FL_MSG (0x03)
Bytes 1-8: Message data contents
Byte 9: CR

END_MESSAGE:
Same as BEGIN_MESSAGE except FL_END_MSG (0x04) marker replaces FL_BEG_MSG.

This messaging scheme is implemented both in the DOM Application and in the dommsg.c routines used by domserver (in fact, dommsg.c is based on the Application code). The messaging protocol has the advantage that it is relatively straightforward to implement and aids in the identification of transmission errors.

4.2.4 Domserver Internals Outline

Domserver has to do many things at once. A single copy of the program controls all 8 DOMs simultaneously. It also receives commands from the executive over the network. And it has to report data collected from each DOM to the RAPCal program for further processing. These tasks must be executed concurrently, with various tasks sharing some of the same data structures.

This sort of situation calls for a multithreaded design. The Linux implementation of POSIX threads (Pthreads) is used for domserver. After some preparatory work, domserver creates a number of threads to handle its various tasks. Mutexes and condition variables are used to prevent race conditions between the different threads, and to allow a thread to take an appropriate action when a different thread signals that it is time to do so. See the reference by Nichols et. al., in the Bibliography, for more information on programming with Pthreads.

4.2.5 Run Control Model - Domserver and Domexec Interactions

Domexec is the program the user uses to interact with the system, although domserver does the bulk of the work of the system. The primary function of domexec is to change the state of domserver. Domserver can be in one of three states, each of which are accessible only from the previous or next state:

1. IDLE: Voltages and other detector properties not set, no data being collected
2. READY: Voltages and other properties are set, but still no data being collected
3. RUNNING: Voltages and detector properties set, data being collected.

The user can start a run if the detector is in IDLE state, but domexec will bring domserver to READY state first. Stopping a run changes the system from RUNNING back to READY, and turning the detector off puts it back in IDLE state.

Another function of domserver is to interact with the DOM database, created using domtest, and pull the appropriate high voltage, discriminator threshold, and other values from the database, passing them on to domserver so that the system can be put in the READY state.

See Section 6, "String 18 Operations in Detail," for more information.

4.2.6 Description of Domserver Threads

Domserver consists of a main thread, which does preparatory work, creates several daughter threads, and simply waits for them to finish (they normally never do!). Several of these threads listen on their own network port for connections from domexec or another external program.

These daughter threads are as follows:

- Character connection handlers (8 copies). These threads listen on port 4000 - 4007 (one thread per port) for connections from programs which expect character-based, rather than message-based, communications to a DOM. When a program such as domtalk connects on one of these ports, data from that program is sent directly to the DOM, and response data from the DOM are sent back to the connecting program.
- Message connection handlers (8 copies, **currently not used**). These threads listen on ports 4010-4017 for programs which expect message-based communication with the DOMs, where domserver handles the packetization and depacketization and the connecting program need only worry about message type, subtype and message contents.
- Data connection handlers (8 copies). These threads listen on ports 4020 - 4027 for connections from programs such as RAPCal which expect to receive physics data from the DOMs. When domserver is in RUNNING mode, any PMT or time calibration data is sent to connecting programs by this thread.

- Web connection handler (1 copy). Programs such as CGI scripts linked into Web pages connect to port 4100 to get status information from domserver. This feature exists in a prototype form only.
- Control connection thread (1 copy). Listens on port 4020. When a connection comes in (typically, from domexec), this thread accepts byte commands from the connecting program and takes the appropriate action. Example commands: start a run; prepare a particular channel for running; get status information for a channel.
- Syncserver connection thread (1 copy). Listens on port 3666. Similar to control connection thread but kept distinct for backward compatibility with older scripts. Handles various DOMCOM FPGA operations.
- Channel control thread (8 copies). This thread acts according to the commands received by the control connection thread, and takes care of message transactions to and from the DOMs. All the details of run handling are dealt with in this thread. Data is collected from the DOMs, and periodic time calibration and monitoring operations are carried out. When PMT or time calibration data are successfully collected, this thread then signals the appropriate data connection handler to send the data on its socket to RAPCal.

4.2.7 Running Domserver

Domserver runs automatically at system boot time by the script installed in /etc/init.d/tbrc (release directory). A superuser can also run it by hand. The following command line options are available:

```
# domserver -h
Usage: domserver
   -h        this message
   -s        skip initial loading of FPGAs
   -f <file> Load FPGA file <file> into DOMCOM boards
   -o <chan> omit domcom channel <chan> (e.g. -o 0234
                        omits channels 0, 2, 3, 4)
```

Unless otherwise directed, domserver will load the default DOMCOM FPGA design (defined as "domcom_12.jam" in the variable "default_fpga_file" in domserver.c) in all DOMCOM boards. The -o option is useful if there are dead or non-instrumented DOMCOM channels (though domserver will not crash if a channel is not operational). The -s option skips the FPGA load phase completely. The -f option allows the user to substitute a different FPGA design file.

4.2.8 Debugging Domserver

As in the case of the device driver, domserver sends diagnostic output to the system log on each DOMCOM PC. Watching (using cat, more or tail) /var/log/messages will show program output as it is generated. You must have superuser privileges to look at the system log.

4.2.9 Running the Executive

To run the executive, type "domexec." A text menu is displayed, along with the contents of the current DOM database:

```
$$ domexec
************************************************
*                                              *
*           Welcome to  d o m e x e c          *
*             by John Jacobsen, LBNL           *
*                                              *
************************************************
SYSTEM SETUP
Open database... ok.
```

```
List of Available DOMs (with associated DOMCOM PCs)..................

 DOM                              DOWN  DEF. DEF. SPE          ATWD ATWD LC   LC
 ID    ADDRESS(DOMCOM:PORT:ID)  DELAY HV   RAT. THR  STATE   MSK1 MSK2 MASK WNDW
 ___ ____________________________ _____ ____ ____ ___  ______ ____ ____ ____ ____
 1045        tbdaq-6:4000(tb0)  0.000    0 0.73 130  ACTIVE 0x03 0x02 0xc1 0xff

Opening control connection to domserver tbdaq-6.lbl.gov...ok.
Opening syncserver connection to domserver tbdaq-6.lbl.gov...ok.


MAIN MENU               ESC->back
_____________________________________________________________________________

        B: Begin a data-taking run.
        E: End a data-taking run.
        P: Turn detector on (HV on all DOMs, etc.)
        O: Turn detector off (HV off, etc.)
        S: Show status of system.

        L: List DOM Database.

        V: Verify connection to domservers (control and sync threads).
        R: Resynchronize DOMCOM clocks.
        Q: Exit.
```

In this example, only one DOM is defined in the database, and rather than the five DOMCOM PCs at the Pole, a single domserver (tbdaq-6.lbl.gov) was defined here by changing the array `@domserver_names` in domexec (normally, this array contains the names of the DOMCOM PCs at the Pole).

At this point, the user could turn on the DOMs (option "P"), start a run (option "B", which implies an intermediate step of turning on the DOMs), or show the status of the system as a whole, including detailed diagnostic information for each channel.

Domexec also allows one to perform operations automatically on the command line, bypassing the primary menu:

```
$$ domexec -status
************************************************
*                                              *
*           Welcome to  d o m e x e c          *
*             by John Jacobsen, LBNL           *
*                                              *
************************************************
SYSTEM SETUP
Open database... ok.

List of Available DOMs (with associated DOMCOM PCs)..................

 DOM                              DOWN  DEF. DEF. SPE          ATWD ATWD LC   LC
 ID    ADDRESS(DOMCOM:PORT:ID)  DELAY HV   RAT. THR  STATE   MSK1 MSK2 MASK WNDW
 ___ ____________________________ _____ ____ ____ ___  ______ ____ ____ ____ ____
 1045        tbdaq-6:4000(tb0)  0.000    0 0.73 130  ACTIVE 0x03 0x02 0xc1 0xff

Opening control connection to domserver tbdaq-6.lbl.gov...ok.
Opening syncserver connection to domserver tbdaq-6.lbl.gov...ok.
DOMserver tbdaq-6.lbl.gov: STATUS READY
DOMs on this DOMCOM PC (also in DOM database):
DOMCOM 0 DOM 1045 moni status: OK
* Monitoring operation succeeded:
  Monitoring status for DOMCOM channel 0:
```

```
DOMCOM FPGA STATUS: LOADED ... LAST LATCHED TIME 0
DOMCOM SYSTEM TIME 1021688191
DOMCOM CLOCK TIMES (2619693581, 64)

DOM is POWERED ON
DOMCOM ADCs 2504(2500) 3329(3300) 2496(2500) 2530(2500) 1(0) 2502(2500)
            1000(1000)
DOM Current 47.865234375 mA
DOMID 1045 ... ADCs 2498 3273 2506 1629 2820 1
                DYNODE 1 V  ANODE 1 V   TOTAL 2 V
DOM FPGA STATUS: LOADED V.0.4
```

This particular use of domexec provides a simple monitoring solution that could be run by of the system's cron program at regular intervals to track the health of the system.

Domexec -h shows the complete list of command-line switches available.

4.3 ADDITIONAL PROGRAMS

The following programs are available for configuring DOM databases, testing DOMs outside of the context of normal operation, and various other system troubleshooting tasks.

4.3.1 Testing DOMs and configuring DOM databases: Domtest

In addition to domexec and domserver, **domtest** is the program the most likely to be used in the normal operation of String 18. Domtest is a large, general purpose testing program which was written to exercise most of the available functions of the DOMs running their Application programs. It is also the program which is used to change configuration information about each channel, such as its location (DOMCOM PC and channel number), high voltage, local coincidence mask setting and single photo-electron threshold.

In order to communicate with a DOM, domtest requires that domserver be running on one or more DOMCOM PCs. It uses domserver's character connection handler thread to send data to and from the DOM. Domtest uses its own message encoder/decoder routines (the DOMSet.pm Perl package), rather than domserver's, since it predates domserver. It also uses DOMTBControl.pm to connect to the syncserver thread in domserver, to carry out DOMCOM FPGA functions.

Domtest is written in Perl and requires several packages throughout the domsoft tree. The program itself and most of the required packages are in the directory domsoft/src/domio, with two additional packages for communicating with domserver in domsoft/src/portio. Installing the software according to Section 5.3, *Installation*, will take care of all the dependencies.

To run domtest, type "domtest" at the Unix shell prompt. You will be asked the location of a DOM database, and where you would like domtest to log its output (in additional to the interactive output sent to the terminal, domtest also logs HTML versions of its output in a separate file for each DOM in the database).

Domtest presents the user with a variety of menus. For example, the top level menu is as follows:

```
AVAILABLE DOM FUNCTIONS         ESC->back
_____________________________________________________________________
        S: Select, list, update or configure the current list of DOMs
        D: DAC Functions
        A: ADC Functions
        H: High Voltage Functions
        F: FPGA Functions
        T: Test Board Functions
        L: Flasher Functions
        O: Other Functions
        Q: Exit domtest
```

Commands are chosen by typing a single keystroke (no return key needed). The top level menu commands present sub-menus of related commands. The escape key leaves the current menu and goes to the higher-level menu (or quits the program is at the top level).

4.3.1.1 Domtest Example: Adding a DOM to your Database

The following example shows how one adds a DOM to a DOM database, effectively creating a new database in the default location.

In all the examples, text in bold italics is typed by the user; non-bold text is output from the program. <CR> or <ESC> means the return or escape key is pressed; menus chosen by single keystrokes are highlighted.

```
$$ domtest

Welcome to /usr/local/dom/bin/domtest (Author: J. Jacobsen, LBNL)
Run by jacobsen on Fri May 17 14:52:24 2002.
_____________________________________________________________________________
_

This program ties up whichever DOMs you access, so please don't leave it
running.
Please enter a directory where DOM test log files are kept
[/usr/local/dom/logs]: <CR>
Please enter a directory where the DOM database is kept (the edit copy)
[/usr/local/dom/db/edit]: <CR>


AVAILABLE DOM FUNCTIONS         ESC->back
_____________________________________________________________________________
        S: Select, list, update or configure the current list of DOMs
        D: DAC Functions
        A: ADC Functions
        H: High Voltage Functions
        F: DOM FPGA Functions
        T: DOMCOM / Test Board Functions
        L: Flasher Functions
        O: Other Functions
        Q: Exit domtest
```

S

```
Select, list, update or configure the current list of DOMs

Please choose one of :

    (L)ist current DOM configuration database
    (V)erify that active DOMs are online
    (P)robe for more DOMs on terminal servers
    (C)hange downgoing delay constant for a DOM
    (T) Set SPE trigger thresh., ATWD and Local Coinc. control masks
    (D)elete one or more DOMs from the database
    (A)ctivate a DOM
    (N) Deactivate one or more DOMs (remove from communications list)
    (S)ubstitute an existing DOM for another one
    (E) Associate a test board ID with a DOM channel
        (for DOMs connected to test boards only!)
    (Q) Change test DAQ address in database
    (I)nteract with the DOMs
```

P

```
Terminal server address [tbdaq-6.lbl.gov]: tbdaq-6.lbl.gov<CR>
Port number (range ok, e.g. 4010-4024) [4000]: 4000<CR>
Got a good DOM at tbdaq-6.lbl.gov:4000: ID is 1045.
Opening DOMtestLog_ID1045trial001.html
Associate port 4000 with DOMCOM board 0 [yes]? <CR>
Probe for more DOMs? [y] n<CR>
```

Now we can look and see if the DOM shows up in the database:

```
Please choose one of :

    (L)ist current DOM configuration database
    (V)erify that active DOMs are online
    (P)robe for more DOMs on terminal servers
    (C)hange downgoing delay constant for a DOM
    (T) Set SPE trigger thresh., ATWD and Local Coinc. control masks
    (D)elete one or more DOMs from the database
    (A)ctivate a DOM
    (N) Deactivate one or more DOMs (remove from communications list)
    (S)ubstitute an existing DOM for another one
    (E) Associate a test board ID with a DOM channel
        (for DOMs connected to test boards only!)
    (Q) Change test DAQ address in database
    (I)nteract with the DOMs
```

L

```
List of Available DOMs (with associated DOMCOM PCs)..................

 DOM                              DOWN  DEF. DEF. SPE          ATWD ATWD LC   LC
 ID    ADDRESS(DOMCOM:PORT:ID)    DELAY HV   RAT. THR  STATE   MSK1 MSK2 MASK WNDW
 ____ ____________________________ _____ ____ ____ ___  _______ ____ ____ ____ ____
 1045         tbdaq-6:4000(tb0)  0.000    0 0.73 130  ACTIVE 0x03 0x02 0xc1 0xff

Please choose one of :

    (L)ist current DOM configuration database
    (V)erify that active DOMs are online
    (P)robe for more DOMs on terminal servers
    (C)hange downgoing delay constant for a DOM
    (T) Set SPE trigger thresh., ATWD and Local Coinc. control masks
    (D)elete one or more DOMs from the database
    (A)ctivate a DOM
    (N) Deactivate one or more DOMs (remove from communications list)
    (S)ubstitute an existing DOM for another one
    (E) Associate a test board ID with a DOM channel
        (for DOMs connected to test boards only!)
    (Q) Change test DAQ address in database
    (I)nteract with the DOMs
```

<ESC>

```
AVAILABLE DOM FUNCTIONS         ESC->back
_____________________________________________________________________________
        S: Select, list, update or configure the current list of DOMs
        D: DAC Functions
        A: ADC Functions
        H: High Voltage Functions
        F: DOM FPGA Functions
        T: DOMCOM / Test Board Functions
        L: Flasher Functions
        O: Other Functions
        Q: Exit domtest
```

<ESC>

```
Testing sequence ended on Fri May 17 14:56:38 2002.
$$
```

The DOM with ID 1045, on DOMCOM Channel 0, on DOMCOM PC tbdaq-6.lbl.gov, is now ready for use by domexec (though the correct high-voltage and other settings are not yet set; this can be done in the "H: High Voltage Functions" and "S: Select, list, update or configure…" menus). One could also continue using domtest and exercise various functions of this DOM.

4.3.2 Talking to DOMs in boot mode: Domtalk

While most interactions with the DOMs occur when they are running their Application program, there are occasions when one needs to talk to the lower-level boot program, domboot. For example, to load a new application program into the DOM's flash memory, change the default DOM FPGA design to be loaded, or simply see if the DOM is alive and working at all, you will want to use the **domtalk** program to run the appropriate domboot commands.

Domboot runs automatically whenever the DOM is power-cycled or rebooted from within the Application. To talk to a DOM connected on a specific channel (0-7) on a given DOMCOM PC (say, tbdaq-6.lbl.gov), type "domtalk tbdaq-6.lbl.gov 0". This connects you directly to the DOM through the domserver program discussed above (character connection handler thread).

When you power on a DOM that you are communicating to using domtalk, you should see a message similar to the following:

```
Welcome to DOMBOOT release (CRC Enabled) of 15-Nov-1999

CRC table initialized...

Timeout value set to 30 seconds...
```

Domboot automatically starts up the Application after a preconfigurable number of seconds have elapsed. To interrupt this, type carriage return immediately, and you should see the following prompt:

```
Domboot 1.16 DOM 16  Enter command (? for menu):
```

With one exception, the keystrokes you type while running domtalk are sent to the DOM boot program. The exception is the escape sequence "Control-]". The escape sequence gives you a series of options:

```
c: Continue  s: Send file  b: Change baud rate  q: Quit
B: Boot DOM Application
```

With the next keystroke, you can tell domtalk what you want to do. The "Send file" option is used when sending a flash file to the DOM - do not do this unless you have instructed the DOM Boot program to accept a file! Otherwise you run the risk of having the DOM interpret the file you send as a sequence of commands which can have bad side effects such as the loss of data in the DOM flash file system. When you select "send file," you can use tab-completion to list and complete file names, much like in newer Unix shells like `tcsh`. Flash files must be prepared in the correct format using the makeflash.pl or the makeflashcrc.pl script in the directory domsoft/src/boot.

The other options are more or less self-explanatory.

This is not a manual for the DOM Boot program, and there currently is none, but most of the commands you will encounter while interacting with the DOMs using domtalk are self-explanatory.

Domtalk is written in Perl and lives in the domsoft/src/domio directory in the CVS tree.

4.3.2.1 Domtalk Example: Uploading a New FPGA Design to a DOM

In this example, we load an FPGA design file into the flash file system of a DOM. We assume we have prepared an FPGA design file prepared with a .fl extension (flash header bytes added using makeflash.pl in domsoft/src/boot). We further assume the DOM is on the first DOMCOM channel of DOMCOM PC tbdaq-6.lbl.gov.

First we connect with the DOM by firing up domtalk on `rust.lbl.gov`, connecting to DOMCOM channel 0 on tbdaq-6:

```
$$ domtalk tbdaq-6.lbl.gov 0
Welcome to /usr/bin/domtalk V0.2, by John Jacobsen / LBNL.
Trying to create socket ...OK.
Trying to talk to board domtest.lbl.gov:3001... press <return> to wake up
DOM.
<CR>
Domboot 1.16 DOM 16  Enter command (? for menu):
```

It is assumed that the DOM is already in boot mode or power-cycled just before return is pressed, above. We now list the contents of the flash file system using the "l" command:

```
Domboot 1.16 DOM 16  Enter command (? for menu): l

Filename:      dom_hv_intg_application
File ID:       4
Major Version: 0
Minor Version: 0
File Type:     Application
Numb. Sectors: 2
Start Offset:  0x00020080
File Size:     240684 (0x0003ac2c)

Filename:      dom_test6
File ID:       4
Major Version: 0
Minor Version: 0
File Type:     FPGA design
Numb. Sectors: 1
Start Offset:  0x00060080
File Size:     45705 (0x0000b289)
```

Now we upload the FPGA file:

```
Domboot 1.16 DOM 16  Enter command (? for menu): u
Enter name of file you will upload:
dom_fpga_test
Enter the file ID, major version, minor version:
8 2 1
Enter the file type (0 = unknown, 1 = FPGA design, 2 = routine, 3 =
application):
1
Send the flash file...
<Control-]>
c: Continue  s: Send file  b: Change baud rate  q: Quit
B: Boot DOM Application

s
Type the name of the file you want.  (Use the TAB key
for file completion / directory listings) : dom_fpga_test<TAB>
Unique match: ./dom_fpga_test.fl

./dom_fpga_test.fl<CR>
Sending file ./dom_fpga_test.fl now (61735 bytes)...
```

```
Done sending file!

c: Continue  s: Send file  b: Change baud rate  q: Quit
B: Boot DOM Application

c
Continuing... press RETURN to get DOM prompt....
Sent = 0x5c50dd64 -- Calculated = 0x5c50dd64
Sent = 0xebfd2ab6 -- Calculated = 0xebfd2ab6
Sent = 0x571bbd85 -- Calculated = 0x571bbd85
…. (some checksums omitted for brevity) ….
Sent = 0x361010a6 -- Calculated = 0x361010a6
Sent = 0x3873a835 -- Calculated = 0x3873a835
Done.  Wrote 61483 bytes, reached memory location 1018f0ab.
61483 bytes were loaded.
```

Now the FPGA design file must be programmed into the flash file system from the DOM's RAM. The "f" command does this:

```
Domboot 1.16 DOM 16  Enter command (? for menu): f
Blowing RAM image into flash...
Sector mask = 0xffffc80f...
First sector used for the new file will be sector 11
Flash ID = 00010001 22492249 00000000
Done programming RAM buffer into flash.
```

The "l" command can now be used again to verify that the image has been added to the flash file system. This is left as an exercise to the reader. To quit domtalk, type the escape combination "Control-]", followed by "q."

Please note that uploading an FPGA file to the flash file system is *not* the same as configuring the actual FPGA hardware using that file. Typically, the configuration of an FPGA is done by domtest, or, more commonly, automatically on startup according to the domboot preferences (which can be changed using the "P" command in domboot).

4.3.3 A simple probe for working DOMs: Domprobe

It is sometimes desirable to simply see if a DOM is up and running the Application. To do this, use the **domprobe** command. For example,

```
$$ domprobe tbdaq-6 0
Got a good DOM (1045), attached to DOMCOM PC tbdaq-6, port 4000.
DOM 1045 at tbdaq-6:4000 : FPGA status -> LOADED
  Major version number: 0
  Minor version number: 4
  Board ID: 0
  FPGA ID: 1
  FPGA file name: DOM 1
```

If there is some problem (DOM not powered on or not working, DOM in boot mode rather than application, DOMCOM FPGA not loaded, DOMCOM PC not reachable, etc.), domprobe will complain that it can't reach a working DOM Application on that particular DOMCOM channel.

4.3.4 Powering DOMs on and off, and loading DOMCOM FPGAs: Domcom

The program called **domcom** allows one to change the power state of various DOMs (turn on, turn off, or power-cycle). It also has several other functions: it allows one to load a new FPGA design into the DOMCOM cards (domserver also does this automatically, unless explicitly told not to with the appropriate

command-line switch); it allows one to read off the eight ADC channels on the card, to measure, for example, the power consumption of the DOM; and finally, to resynchronize all the DOMCOM oscillator counters on the next 1 PPS signal from the clock distribution system, so that each oscillator reflects the proper universal time. This resync operation has to be done before collecting physics data with domexec, though domexec also has the capability to perform a clock resync.

The full list of available commands can be gotten by typing "domcom -h". An interactive menu of commands can also be gotten by typing "domcom *pc channel* where *pc* is one of tbdaq-1.spole.gov, … and *channel* is 0-7.

4.3.4.1 DOMCOM Examples

To resynchronize ALL the DOMCOM oscillators,

```
$$ domcom tbdaq-1 tbdaq-2 tbdaq-3 tbdaq-4 tbdaq-5 0-7 resync
```

Do this on string18.spole.gov so that the .spole.gov domain is assumed for tbdaq-1, etc.

To power cycle a single DOM (on channel 4) of tbdaq-2,

```
$$ domcom tbdaq-2 4 reboot
```

Note: DOMCOM channels are powered ON by default.

4.3.5 Capturing DOM data in absence of RAPCal: SimRAPCal

The task of the String 18 Data Acquisition is not simply to unload raw data from the DOMs, but to build meaningful triggers of detected muons for physics analysis. Photomultiplier pulses from the DOM are stamped with the time of the local (DOM) oscillator, which must be converted into the global time using the periodically-acquired RAP time calibration data. To this end, the RAPCal program was proposed and nearly fully implemented by Kael Hanson and Doug Cowen at the University of Pennsylvania, but as of this writing the software has not been integrated with domserver and domexec. Downstream of RAPCal, an event trigger called EBTrig is meant to search for clusters of hits in space and time, and to pass on these "software trigger" events to the AMANDA disk cache for transmission via satellite.

In absence of full RAPCal/EBTrig integration, the program SimRAPCal.pl was written to consume data from domserver and redirect it to disk. For example, if the DOMCOM PC tbdaq-1.spole.gov has eight working channels, to capture data to disk for all those channels, type:

```
$$ SimRAPCal.pl tbdaq-1.spole.gov 0 > data_file_0.dat &
$$ SimRAPCal.pl tbdaq-1.spole.gov 1 > data_file_1.dat &
.
.
.
$$ SimRAPCal.pl tbdaq-1.spole.gov 7 > data_file_7.dat &
```

Note that SimRAPCal.pl can be run anywhere on the network but was designed to be run on string18.spole.gov. These eight copies of SimRAPCal will run continuously, redirecting their outputs to the appropriate data files, until they are killed, domserver restarts, or the system reboots. These data files can then be analyzed offline.

5 Acquiring and Building the Software

5.1 How to use the domsoft Repository

The CVS repository for the String 18 software lives on rust.lbl.gov; to use it, you must have an account on the machine. To check out the repository on rust.lbl.gov, add the following line to your .cshrc or .login:

```
setenv CVSROOT /usr/local/cvsroot
```

Then,

```
cvs checkout domsoft
```

On another Unix machine:

```
setenv CVSROOT myname@rust.lbl.gov:/usr/local/csvroot
setenv CVS_RSH ssh
cvs checkout domsoft
```

For remote CVS operations you will have to give your password each time you check out, update or commit new software.

5.2 How to Compile the Software

The software tree is organized into makefiles; building it should be as simple as changing your working directory to the domsoft/src directory, and typing "make". You can also type "make" inside the various subdirectories of domsoft/src if you want to build only the software inside that directory.

5.3 Installation

To install the currently compiled version of the software, type "make install" in the domsoft/src directory. You can install the contents of a subdirectory (e.g., domsoft/src/driver) by changing to that directory and typing "make install" from there. You should be root before typing "make install."

Installing the programs copies them to /usr/local/dom/bin, with accessory libraries (such as the Perl packages used by the scripts) living in /usr/local/dom/lib. Make sure the former directory is in your path if you are going to use these programs outside of the "dom" account.

6 String 18 Operations in Detail

This section consists of a series of notes which present a detailed outline of what happens during different phases of operation of the string. A few of these steps are not completely implemented in the system, but the outline as a whole provides a useful guide.

6.1 COMMUNICATION CHANNELS, ENUMERATED

The following channels of information are relevant. Each has its own brand of "message," so the term message is vague in this context (unless otherwise specified, "message" refers to the packetized message format used to communicate with the DOM Application). The diagram at the end of the document may be helpful for the visualization of the system as a whole.

CC 1. domserver channel control thread to DOM application (DOM communications channel).
CC 2. executive to domserver control connection thread [port 4200] (DOMCOM control channel).
CC 3. executive to trigger/event builder [port 4201] (EBTrig) (event builder control channel)
CC 4. domserver data connection thread to RAPCal [port 4020-4027] (uncalibrated data channel)
CC 5. RAPCal to event builder [port undefined] (calibrated data channel)

In addition to the above channels, which are needed for normal operation of the string, the following channels are needed for testing, configuring, or backwards compatibility with existing software:
CC 6. domtalk to DOM boot program, via domserver character connection thread (raw character channel). Older/existing Perl programs such as domtest use the raw character channel as well, encoding the messages directly rather than going through Channel 2. This channel lives on each DOMCOM PC, ports 4000-4007.
CC 7. syncserver message channel (used by the programs domcom, domtest, domlogger). This channel lives at Port 3666.
CC 8. FastDOMMsg channel (used by pre-domserver Perl programs, which sent message "summaries" to a program "messageserver" running in the DOM; this "messageserver" then sent the fully-packetized messages on Channel 1 to the DOMs). This channel lives on each DOMCOM PC, ports 4010-4017. This functionality is deprecated.

6.2 STRING 18 PHASES OF OPERATION

Items flagged in **Bold** still need to be fully implemented.

1. HARDWARE POWER-UP AND BOOT

Power on each of 5 DOMCOM PCs (tbdaq-1 through -5). Linux boots. Init script "tbrc" starts 8 RAPCal programs and one domserver program, and loads the kernel device driver for the DOMCOM boards. Domserver starts the following threads:

- Data connection (x 8) (for RAPCal)
- Control Connection (for domexec)
- Syncserver (for domexec; deprecated, for backward compatibility w/ Perl scripts)
- Channel Control (x 8) (one thread to control each channel)
- Message I/O (x 8) (deprecated, for backward compatibility w/ Perl scripts)
- Character I/O (x 8) (for talking w/ DOMs in boot mode, and for messaging w/ older Perl scripts)
- Web info (for getting status of system through a Web browser)

String18 PC power-on: Linux boots. **Init script "ebrc" starts EBTrig and RAPCal daemons. RAPCal establishes communication with domserver's data source thread on CC 4.**
Domserver loads default FPGA in DOMCOM boards.
Domserver opens/initializes the DOMCOM device driver.

A startup script on tbdaq-5 loads the clock distribution system FPGA through the jamplayer (currently done by hand).

HP Power supply on - DOMs boot applications and load FPGAs (this takes ~ 100 seconds).

2. EXECUTIVE STARTUP

When data taking is to begin, the executive is run by the "dom" account on string18, by typing "domexec."

Executive reads database of DOMs. Database was produced by domtest and stores DOMCOM addresses, HV settings, local coincidence settings, etc.

Executive connects to EBTrig (CC 3).

Executive tells EBTrig which addresses to use for incoming data.

Executive connects to all 5 domserver's control threads (CC 2).

If the user desires, executive tells all DOMCOM FPGAs to synchronize their clocks to the next 1pps signal from the GPS clock; domserver reports the correct values to RAPCal.

3. DETECTOR INITIALIZATION

Once the executive is started up, it can initialize the DOMCOM hardware and DOMs to prepare for data taking. At the highest level, this consists of messages from domexec to domserver (CC 2).

- Make sure DOMs are booted into application
- Make sure DOM FPGAs are loaded **and identical for all DOMs.**
- **Start fast communications:**
    1. domserver's message I/O thread tells DOM to change speed
    2. DOM changes speed
    3. domserver's syncserver thread tells DOMCOM FPGA to change speed
    4. domserver's message thread issues test message to make sure it worked
    5. domserver's message thread tells DOM if it got the correct reply.
    6. (The double-message at the end is required so that both parties know that the new speed is in force - what happens if fast communications doesn't work?)
- Set DOM local coincidence FPGA registers (DOM dependent)
- Turn on and check DOM high voltage
- Set DOM SPE discriminators
- **Make sure clock distribution system has stabilized.**
- Set ATWD trigger masks
- Time calibration initialization: set appropriate FPGA registers in DOM; set appropriate DAC.

4. BEGIN RUN

Run initialization

Exec tells EBTrig (via CC 3) the following:

- what run number and file name base to use
- how big to make each file in the run
- trigger configuration info (according to Kael) -
    1. N hits in space window of M contiguous DOMs
    2. Time window (nsec)
    3. H, hit threshold above which you only check the time coincidence and not the space coincidence. (This allows you to always trigger when you get 6 hits, say, in the time window, irrespective of their locations on the string. There is physical motivation for this triggering.)

Exec tells domserver control thread via CC 2 to perform N time calibrations. Control thread sends the result to RAPCal via the data source thread using CC 4. RAPCal sends the result to EBTrig on CC 5.

Exec tells domserver control thread to enter "normal" data collection mode.

Control thread loop:

Get PMT data from DOM (as often as possible, when not performing time calibration and monitoring):

- ? domserver control thread decides to retrieve some data.
- ? domserver message thread -(CC 1)-> DOM: give me some data.
- ? DOM reads out lookback memory & sends the data on CC1;
- ? domserver data source thread reports the data to RAPCal on CC 4;
- ? RAPCal reports time-calibrated data to EBTrig on CC 5 (Azriel wants the ability to get the raw data too).

Perform "normal running" time calibration (every 10 seconds):

- ? Domserver message thread -(CC 1)-> DOM: start time calibration sequence.
- ? Domserver syncserver thread disables DOMCOM to DOM communications on CC 1.
- ? Domserver syncserver thread: Issues time tick.
- ? DOM sends upgoing time tick.
- ? Domserver syncserver thread reads out surface ADC waveform and time stamps
- ? Domserver message thread -(CC 1)-> DOM: give me ADC waveform in DOM and time stamps
- ? Domserver syncserver thread enables DOMCOM to DOM communications on CC 1.
- ? Domserver data source thread reports surface/DOM waveforms and surface/DOM time stamps to RAPCal on CC 4.
- ? RAPCal uses this data however it likes! (To correctly time-calibrate data for EBTrig).

Perform "monitoring" (every 100 seconds):

Domserver message thread -(CC 1)-> DOM: get the following information:

- ✍ DOM ID
- ✍ 8 DOM ADC values
- ✍ DOM FPGA status

Domserver also uses the portio interface to get:

- ✍ DOMCOM FPGA status
- ✍ 8 DOMCOM ADC values
- ✍ Power state of the DOM
- ✍ Local 64 bit DOMCOM board time

Finally, domserver gets the system time using the DOMCOM PC's operating system call, time().

These values are all stored in a data structure for domexec or another program to fetch on demand.

5. STOP RUN

Domexec tells domserver to change its state from "RUNNING" to "READY"

Do a final time calibration. Domserver sends results to RAPCals, along with an end-of-run marker.

Exec tells EBTrig to end the run, via a message over the control socket.

domserver, RAPCals keep running and just wait.

NOTE:

Domexec doesn't have to be running after the run has started. Domexec is run only to start a run, to stop a run, or to query the status of each channel and the system as a whole.

6. DETECTOR SHUTDOWN

   Turn off high voltage. Domexec tells domserver via CC 2 to change the state from "READY" to "IDLE". Domserver zeroes out the voltages on each DOM using CC 1.

   Power DOMs off as well? If so, domexec -CC 2-> domserver's syncserver thread to set the appropriate FPGA registers.

7. CLOSEOUT

   This should be a rare occurrence -- normally, you want to leave things powered on. To power off, issue Linux shutdown command to shut down DOMCOM PCs and string18. Power off these PCs and the HP power supply.

7 Bibliography

DOMCOM PC Installation Instructions, John Jacobsen, LBNL, 2001. Information on the installation and configuration of the DOMCOM PCs. Available at http://rust.lbl.gov/~jacobsen

Pthreads Programming, Bradford Nichols et. al., O'Reilly & Associates, 1996.

Linux Device Drivers, Rubini & Corbet, O'Reilly & Associates, 2nd Edition, 2001.

DOMCOM API, Rev. 1.3, Karl-Heinz Sulanke, DESY-Zeuthen, 2001. Official DOMCOM FPGA API used by domserver. http://www-zeuthen.desy.de/~sulanke/Projects/DOMCOM/Doc/Domcom_API.doc

API for the DOM Test Board, Gerald Przybylski, LBNL, 2000-2002. Information on the FPGA registers used by domserver. http://rust.lbl.gov/~gtp/local/testboard_API.htm

8 Appendix

Diagram of the DAQ software components. Items in grey are included in the domserver program.